esprit Se former autrement HONORIS UNITED UNIVERSITIES	Année universitaire : 2025-2026 **Workshops Framework Symfony6** **UP-Web**

Workshop :
Validation d'un formulaire côté serveur

Objectifs

- Valider un formulaire côté serveur.
- Personnaliser un formulaire

Travail demandé :

NB : Dans ce workshop, nous allons travailler avec une seule entité « ***Student*** » comme le montre la figure suivante, avec **Nsc** : le numéro d'inscription

Student
Nsc
Email

Le contenu de cette classe est similaire à celui montré dans la figure suivante :

```php
use App\Entity\Student;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;

class StudentType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add("nsc")
            ->add('email')
            ->add('save',SubmitType::class)

        ;
    }
```

Figure 1: Le contenu du fichier "StudentType.php"

1- Validation côté serveur

a. Ajout des contraintes au niveau de l'entité

```php
use App\Repository\StudentRepository;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Validator\Constraints as Assert;
#[ORM\Entity(repositoryClass: StudentRepository::class)]
class Student
{



    #[ORM\Id]
    #[ORM\Column]
    #[Assert\NotBlank(message:"NSC is required")]
    private ?int $nsc = null;


    #[ORM\Column(length: 255)]
    #[Assert\NotBlank(message:"Email is required")]
    #[Assert\Email(message:"The email '{{ value }}' is not a valid email ")]
    private ?string $email = null;
```

Figure 2: Le contenu du fichier "StudentType.php" avec contraintes

b. Test de la validité du formulaire au niveau du contrôleur

```php
#[Route('/student/add', name: 'student_add')]
public function addStudent(ManagerRegistry $doctrine,Request $req): Response {
    $em = $doctrine->getManager();
    $student = new Student();
    $form = $this->createForm(StudentType::class,$student);
    $form->handleRequest($req);
    if($form->isSubmitted() && $form->isValid()){
        $em->persist($student);
        $em->flush();
        return $this->redirectToRoute('student_add');
    }

    return $this->renderForm('student/add.html.twig',['form'=>$form]);
}
```

Figure 3: Le contenu du fichier "StudentController.php"

2- Personnalisation de l'affichage

```
<h1> Ajouter un étudiant </h1>


{{form_start(form,{'attr': {'novalidate': 'novalidate'}} )}}

<table>
<tr>
    <td> {{ form_label(form.nsc,"Numéro d'inscription")  }} </td>
    <td>  {{ form_widget(form.nsc) }} </td>
    <td> {{ form_errors(form.nsc) }} </td>
    </tr>
<tr>
    <td>  {{ form_label(form.email,"Email") }} </td>
    <td> {{ form_widget(form.email)}} </td>
    <td>  {{ form_errors(form.email) }} </td>
</tr>
</table>

    {{ form_widget(form.save)}}

{{form_end(form)}}
```

Figure 4: Le contenu du fichier « add.html.twig » mis à jour

RQ : Pour afficher les messages d'erreur, il faut désactiver le contrôle de saisie HTML5. Pour ce faire, nous ajouterons cette ligne de code dans le fichier Twig:

```
{{ form_start(form,{'attr': {'novalidate': 'novalidate'}} ) }}
```

Résultat final :

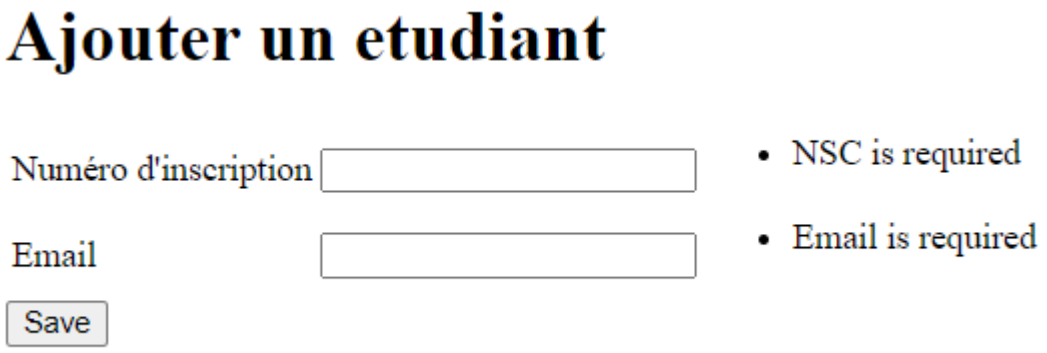

Figure 5: Résultat

Documentation :

1- Validation du formulaire

1) Principe

- Symfony permet l'ajout des contrôles de saisie dans les formulaires.
- La validation des objets (entités ou autres) se réalise avec le composant ***Validator*** de Symfony.
- Ce composant est dédié pour valider les objets selon des contraintes sur le formulaire.
- Pour assurer le fait que les données saisies sont valides, il faut ajouter la fonction **isValid ()**.

2) Syntaxe

- **Utilisation de namespace*:***

Afin de pouvoir utiliser les annotations de validation il faut importer la class Constraints et l'ajouter dans l'entité « **Student** ».

```
use Symfony\Component\Validator\Constraints as Assert;
```

- **La forme simple :**

```
#[Assert\ Contrainte (valeur de l'option par défaut)]
```

Avec:

❑ La `Contrainte`, est utilisée pour affirmer une condition appliquée sur la valeur d'une propriété. Peut-être, par exemple, **`MinLength`**, **NotBlank**, **IsNull**, etc.

❑ La `Valeur` entre parenthèses, qui est la valeur de l'option par défaut. En effet chaque contrainte a plusieurs options, dont une par défaut. Par exemple, l'option par défaut de **`MinLength`** est certainement la valeur de la longueur minimale que l'on veut appliquer.

- **La forme étendue :**

Nous pouvons également utiliser la forme étendue, qui permet de personnaliser la valeur de plusieurs options en même temps, comme indiqué ci-dessous:

```
#[Assert\Contrainte (option1:"valeur1", ...
optionN:"valeurN")]
```

Les différentes options varient d'une contrainte à l'autre.

Ci-après, un exemple avec la contrainte **`Length`** :

- **Exemple :**

```
#[Assert\Length(min:10,message:"Votre mot  de passe ne contient pas {{ limit }} caractères."))]
```

3) Les contraintes

Parmi les contraintes, on peut citer :

contrainte	Définition
Blank/ NotBlank	Vérifie si la valeur d'un champ est une chaîne vide ou null (NotBlank est l'inverse)
IsTrue	Vérifie si la valeur d'un champ est true, 1 ou '1' (pareillement pour IsFalse
Length	Vérifie si la longueur d'un champ se situe entre une valeur minimale et une valeur maximale. Elle accepte plusieurs options : ▪ **min** ▪ **minMessage**: le message d'erreur à afficher si la contrainte min n'est pas respectée. ▪ **max** ▪ **maxMessage**: le message d'erreur à afficher si la contrainte max n'est pas respectée.
url	Vérifie si la valeur est une adresse URL valide
UserPassword	Vérifie que la valeur saisie dans l'input égale à celle de l'utilisateur connecté.
Date / DateTime	Vérifie que la valeur est un objet de type Datetime, ou une chaîne de caractères du type YYYY-MM-DD / YYYY-MM-DD HH:MM:SS.

Positive	Valide qu'une valeur est un nombre positif.

- Nous allons faire le contrôle de saisie de notre formulaire (Voir figure 5) qui doit respecter les contraintes suivantes :

 Le numéro d'inscription doit être un champ obligatoire. Nous allons utiliser la contrainte (**\NotBlank**),

 L'Email soit un champ obligatoire et de type Email alors nous allons utiliser deux contraintes (**\NotBlank, \Email**).

2- Personnalisation de l'affichage

★ ***form_start () :***

- C'est l'équivalent de la balise <form> en HTML
- Accepte deux paramètres :
  1. Le premier est la variable formulaire (passée depuis le contrôleur).
  2. Le deuxième, facultatif, contient l'index « ***attr*** » des paramètres (par exemple nom de la classe bootstrap).

Exemple:

```
{{form_start(form,{'attr':{'class':'form-horizontal'}})}}
```

★ ***form_errors () :***

Elle affiche les erreurs relatives au champ en question (celui passé en argument)

★ ***form_label():***

- Affiche le label HTML d'un élément du formulaire.
- Accepte deux paramètres :

1. Le premier est le champ concerné par le label.
2. Le deuxième, facultatif, est le contenu du label (par défaut, le nom de l'attribut ou la valeur « label » dans le form type.

⇨ **Exemple :**

```
{{ form_label(form.nsc,"Numéro d'inscription") }}
```

★ ***form_widget () :***
Affiche le champ du formulaire lui-même (soit <input>, soit <select>…)

★ ***form_row () :***
form_label + form_errors + form_widget

★ ***form_rest () :***

⇨ Affiche le reste du formulaire (les champs que nous n'avons pas précisé)

★ ***form_end () :***
C'est l'équivalent de la balise fermante </form> en HTML

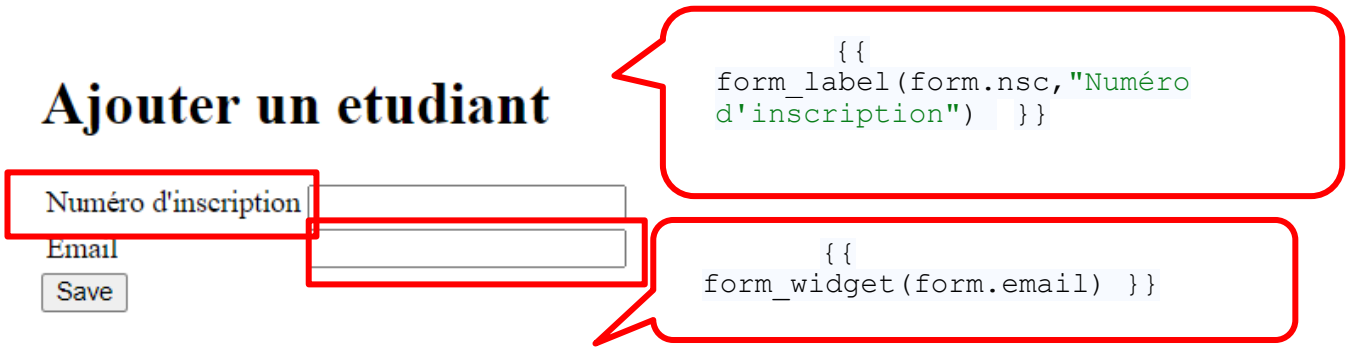

Figure 6: Le rendu personnalisé de la page d'ajout d'un formulaire